

# Faltungsnetze

## Vorlesung 5, Deep Learning

Dozenten: Prof. Dr. M. O. Franz, Prof. Dr. O. Dürr

HTWG Konstanz, Fakultät für Informatik

# Übersicht

1 Moderne CNN-Architekturen

2 Deep Learning mit Faltungsnetzen

3 Moderne CNN-Architekturen

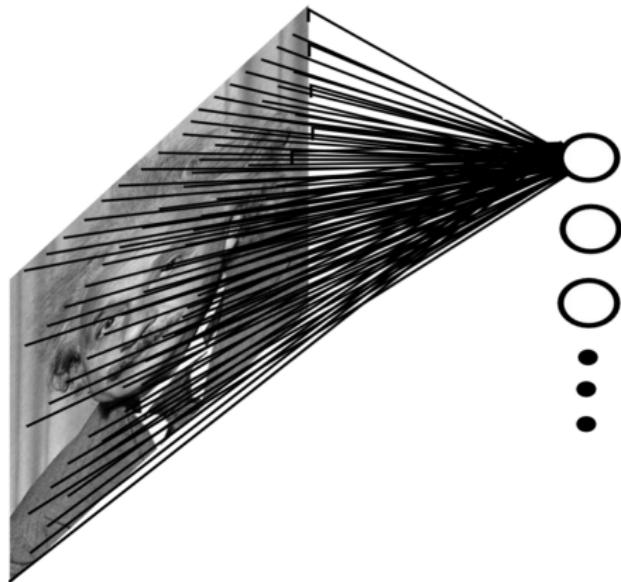
# Übersicht

1 Moderne CNN-Architekturen

2 Deep Learning mit Faltungsnetzen

3 Moderne CNN-Architekturen

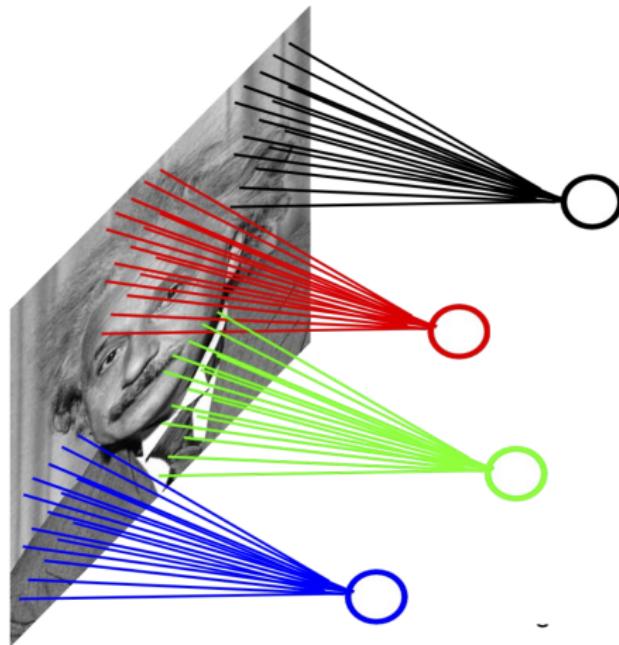
# Gefahr des Overfittings in MLPs



- In einem traditionellen MLP ist jedes Neuron mit allen Neuronen der vorangehenden Schicht verbunden.
- Ein  $200 \times 200$  Eingangsbild mit 40.000 Neuronen in der verdeckten Schicht bräuchte 2 Mrd. Gewichte!
- Es gibt selten genug Trainingsbeispiele, um so viele Parameter festzulegen  $\Rightarrow$  Overfitting.
- Korrelationen im Bild sind i. W. nur auf die direkte Nachbarschaft beschränkt.

[Ranzato, 2014]

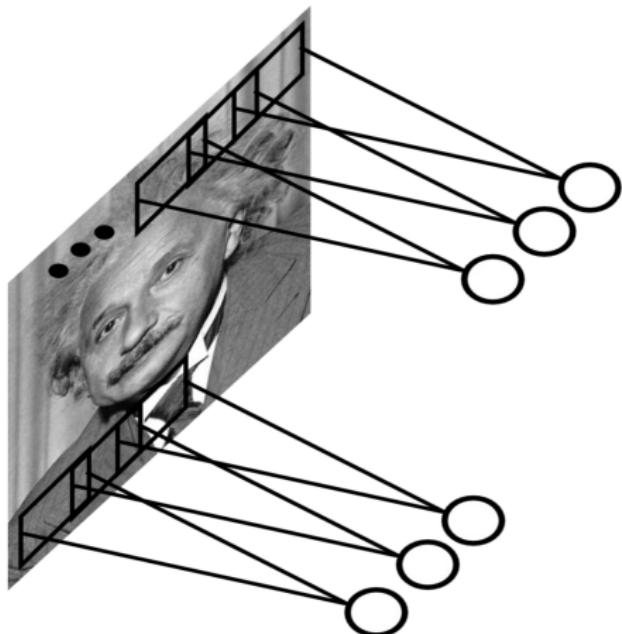
# Möglicher Ansatz: lokal verknüpfte Neuronen



- Ein 200 x 200 Eingangsbild mit 40.000 Neuronen in der verdeckten Schicht und 10 x 10 rezeptivem Feld braucht nur noch 4 Mio. Parameter.
- Funktioniert gut, wenn alle Bilder genau registriert sind.
- Die lokale Statistik in verschiedenen Bildausschnitten ist oft sehr ähnlich. Reicht vielleicht ein einheitlicher Gewichtssatz?

[Ranzato, 2014]

# Faltung durch Neuronen (Faltungsschicht, Convolutional Layer)



[Ranzato, 2014]

- Jedes Neuron der verdeckten Schicht teilt sich seine Gewichte mit allen anderen: nur noch 100 Gewichte!
- Die rezeptiven Felder überlappen sich.
- Entspricht einer Korrelation mit einer **gelernten** Filtermaske.
- Je nach Randbehandlung und Überlappungsgrad der Fenster sind die Faltungsschichten gleich groß oder kleiner als die Eingangsschicht.

# Beispiel Faltungsschicht

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3 <sub>0</sub>	2 <sub>1</sub>	1 <sub>2</sub>	0
0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>0</sub>	1
3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>
0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>0</sub>
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>	3	1
3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>0</sub>	2	3
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0 <sub>0</sub>	1 <sub>1</sub>	3 <sub>2</sub>	1
3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	3
2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1 <sub>0</sub>	3 <sub>1</sub>	1 <sub>2</sub>
3	1	2 <sub>2</sub>	2 <sub>2</sub>	3 <sub>0</sub>
2	0	0 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2 <sub>2</sub>	0 <sub>2</sub>	0 <sub>0</sub>	2	2
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3
2	0 <sub>2</sub>	0 <sub>2</sub>	2 <sub>0</sub>	2
2	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>
2	0	0 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>
2	0	0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>3</sub>

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

[Dumoulin & Visin, 2016]

# Ein Filter sucht nach einem spezifischen lokalen Merkmal



image patch						
0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

filter/kernel: curve detector

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

$$= 6600$$

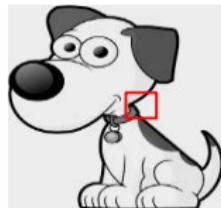


image patch						
0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

filter/kernel: curve detector

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

$$= 0$$

We get a large resulting value if the filter resembles the pattern in the image patch on which the filter was applied.

# Beispiel Faltungsschicht mit 2er-Schritten und Zeropadding

0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0	0	0	0
0 <sub>2</sub>	3 <sub>2</sub>	3 <sub>0</sub>	2	1	0	0
0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0	0
0	3	3 <sub>2</sub>	2 <sub>2</sub>	1 <sub>0</sub>	0	0
0	0	0 <sub>0</sub>	1 <sub>1</sub>	3 <sub>2</sub>	1	0
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1 <sub>2</sub>	0 <sub>2</sub>	0 <sub>0</sub>
0	0	0	1	3 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	1	3	1	0
0 <sub>2</sub>	3 <sub>2</sub>	3 <sub>0</sub>	2	2	3	0
0 <sub>0</sub>	2 <sub>1</sub>	0 <sub>2</sub>	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0 <sub>0</sub>	1 <sub>1</sub>	3 <sub>2</sub>	1	0
0	3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	3	0
0	2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>
0	3	1	2	2 <sub>2</sub>	3 <sub>2</sub>	0 <sub>0</sub>
0	2	0	0	2 <sub>0</sub>	2 <sub>1</sub>	0 <sub>2</sub>
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0 <sub>0</sub>	2 <sub>1</sub>	0 <sub>2</sub>	0	2	2	0
0 <sub>2</sub>	2 <sub>0</sub>	0	0	0	1	0
0 <sub>0</sub>	1 <sub>1</sub>	0 <sub>2</sub>	0	0	0	1

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0	2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	2 <sub>0</sub>	2 <sub>1</sub>	0 <sub>2</sub>
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0 14.0 17.0  
14.0 12.0 12.0  
8.0 10.0 17.0

## Kreuzkorrelation und Faltung

Trotz ihres irreführenden Namens führen Faltungsschichten eigentlich keine Faltung des Bildes  $I$  durch, sondern eine **Kreuzkorrelation** mit einer  $n \times n$ -Filtermaske  $w$ :

$$a(x, y) = I \otimes w = \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} I(x + x', y + y') w(x', y'),$$

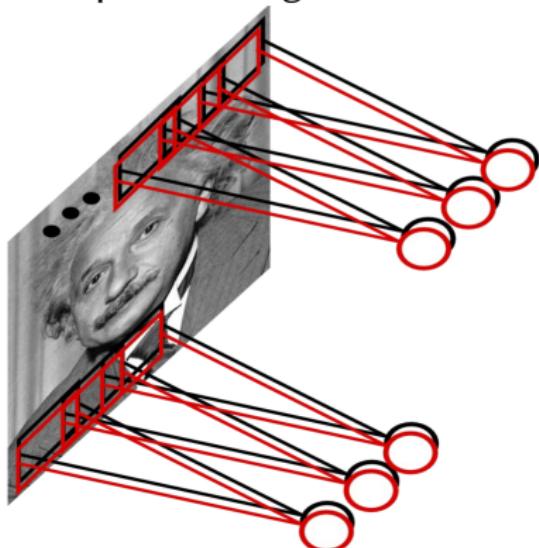
d.h. die Maske wird wie ein Fenster über das Eingangsbild gelegt und punktweise multipliziert. Vgl. hierzu die **Faltung**:

$$\begin{aligned} a(x, y) &= I * w = \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} I(x - x', y - y') w(x', y') \\ &= \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} I(x + x', y + y') w(-x', -y'), \end{aligned}$$

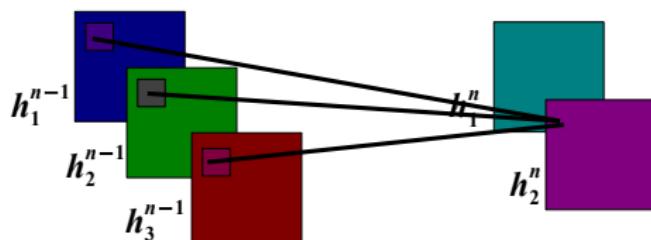
d.h. die Maske wird bei der Faltung zuerst gespiegelt und erst dann korreliert!

# Mehrere Filter und Eingabekanäle

Meist werden mehrere gelernte Filter parallel eingesetzt:

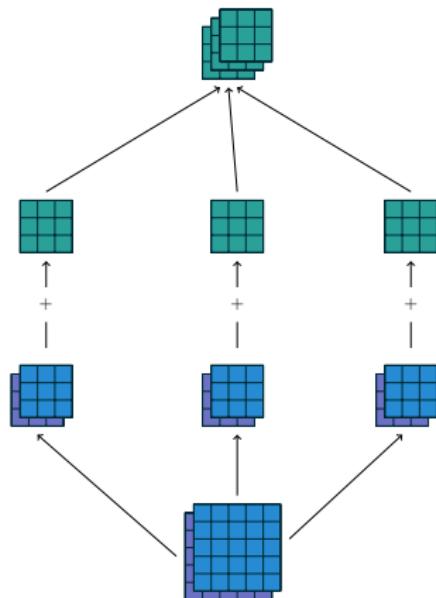


- Bei  $n$  Filtern ist der Output der Faltungsschicht ein Block aus  $n$  *feature maps*.
- Besteht der Input aus mehreren Kanälen (z.B. Farben oder Filterausgaben der vorigen Schicht), haben die Fenster in allen Kanälen die gleiche Größe und Position, aber eigene Gewichte.



[Ranzato, 2014]

## Beispiel: 2 Eingangskanäle, 3 *feature maps*



[Dumoulin & Visin, 2016]

- Jede Faltung erzeugt aus einem Eingangsbild wieder ein Ausgangsbild.
- Das Eingangsbild kann mehrere Kanäle haben, das Ausgangsbild hat so viele Kanäle wie die Faltungsschicht feature maps hat.
- I.A. transformiert eine Faltungsschicht also einen dreidimensionalen Pixelblock in einen neuen dreidimensionalen Pixelblock.

## Faltungsschicht: Notation

Im Gegensatz zu klassischen MLPs sind bei Faltungsnetzen die einzelnen Schichten zweidimensional (bei nur einem Eingangskanal) oder dreidimensional (bei mehreren Eingangskanälen).

Wie üblich hat auch das Faltungsneuron einen Schwellwert und eine nichtlineare Aktivierungsfunktion. Im Netzwerk wird die Faltungsschicht mithilfe von 2D-Indizes  $x, y, x', y'$  als

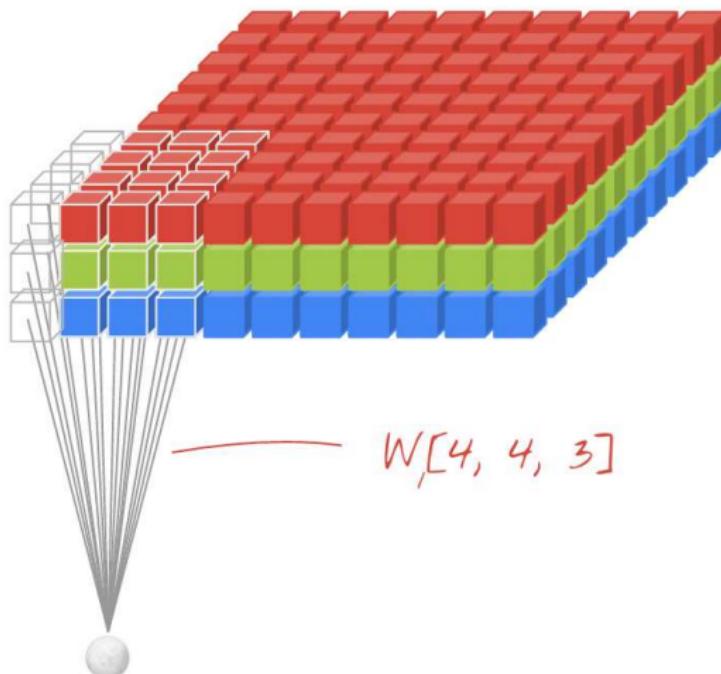
$$a_{x,y}^l = \sigma \left( \sum_{x'} \sum_{y'} a_{x-x', y-y'}^{l-1} w_{x', y'}^l + b^l \right)$$

geschrieben, bzw. als schichtenweise Korrelation mit 3D-Indizes  $x, y, z, x', y'$

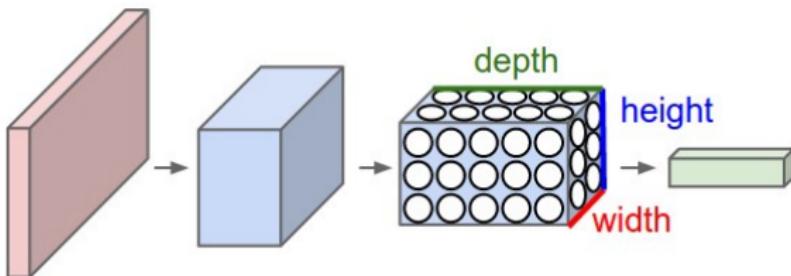
$$a_{x,y}^l = \sigma \left( \sum_{x'} \sum_{y'} \sum_z a_{x-x', y-y', z}^{l-1} w_{x', y', z}^l + b^l \right).$$

In Matrixform:  $a^l = \sigma(a^{l-1} \otimes w^l + b^l)$ .

## Beispiel: Faltung eines mehrkanaligen Bildes



# Tensordarstellung



[<http://cs231n.github.io/convolutional-networks/>]

Die dreidimensionalen Arrays  $w_{x',y',z}^l$  der Gewichte in den Faltungsschichten entsprechen einer Erweiterung des Matrixbegriffs auf mehr als zwei Dimensionen. Diese werden oft (mathematisch unpräzise) **Tensoren** genannt. Breite und Höhe sind die Größe der Filtermaske, die Tiefe die Anzahl der Eingangskanäle. Bei mehreren *feature maps* innerhalb der Schicht ist der Gewichtstensor sogar 4-dimensional.

# Backpropagation für Faltungsschichten

Die Ableitung der Gleichungen für Backpropagation in Faltungsschichten ist deutlich aufwendiger als bei komplett verbundenen Schichten\*. Wir erhalten mit  $z^l = a^{l-1} \otimes w^l + b^l$  für ein einzelnes Faltungsneuron

$$1. \quad \delta^{l-1} = (\delta^l * w^l) \odot \sigma'(z^l)$$

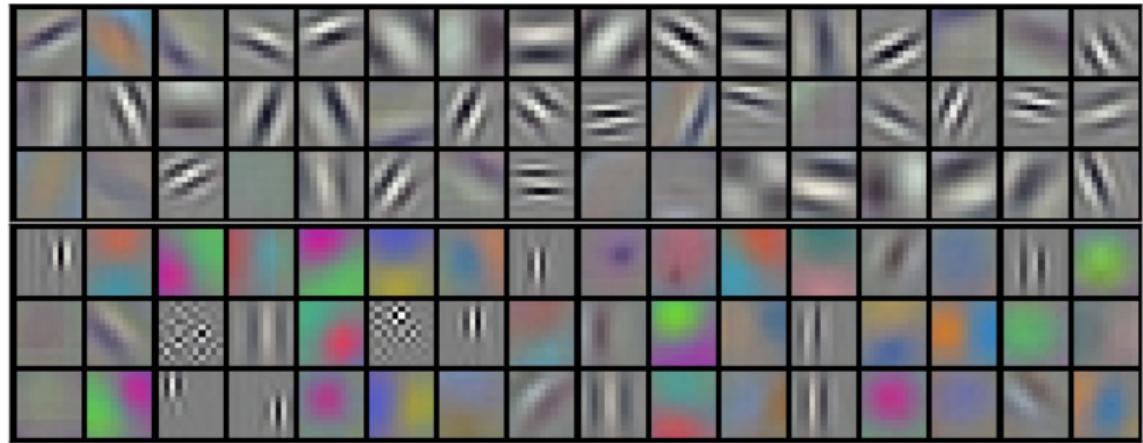
$$2. \quad \frac{\partial C}{\partial w^l} = a^{l-1} \otimes \delta^l$$

$$3. \quad \frac{\partial C}{\partial b^l} = \sum_{ij} \delta_{ij}^l$$

Meist wird der Gradient automatisch innerhalb des verwendeten Frameworks (z.B. Tensorflow oder PyTorch) berechnet.

\* s. z.B. <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>

## Beispiel: gelernte Filter (AlexNet, 2012)



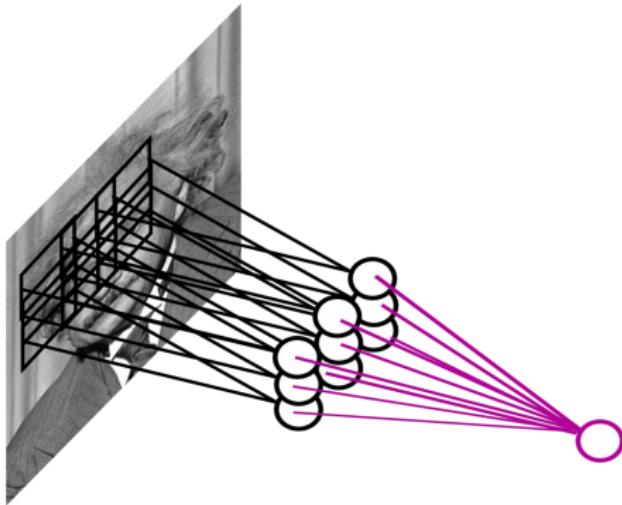
# Übersicht

1 Moderne CNN-Architekturen

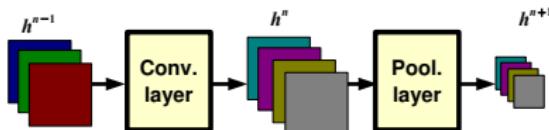
2 Deep Learning mit Faltungsnetzen

3 Moderne CNN-Architekturen

# Pooling-Schichten



[Ranzato, 2014]



- Bei Pooling nimmt man z.B. das Maximum oder den Durchschnitt des Filteroutputs in einer lokalen Nachbarschaft zum Erreichen einer gewissen Translationsinvarianz.
- Durch eine Pooling-Schicht verkleinert sich die Größe des Outputs je nach Schrittweite zwischen den Pools.
- Pooling-Schichten haben keine trainierbaren Gewichte.

# Unterschiedliche Poolingtypen

Der Output wird um den Faktor  $m$  in x-Richtung und um  $n$  in y-Richtung verkleinert, die Anzahl der *feature maps* bleibt gleich.  
 $\mathcal{N}_{x/m, y/n}$  ist der Pool des Neurons.

- Max-Pooling:

$$a_{x/m, y/n}^l = \max \left\{ a_{x,y}^{l-1} \mid x, y \in \mathcal{N}_{x/m, y/n} \right\}$$

- Durchschnittspooling:

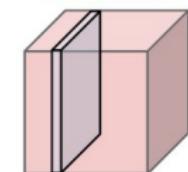
$$a_{x/m, y/n}^l = \frac{1}{mn} \sum_{x,y \in \mathcal{N}_{x/m, y/n}} a_{x,y}^{l-1}$$

- L2-Pooling:

$$a_{x/m, y/n}^l = \sqrt{\sum_{x,y \in \mathcal{N}_{x/m, y/n}} (a_{x,y}^{l-1})^2}$$

# Beispiel Max-Pooling

224x224x64



pool

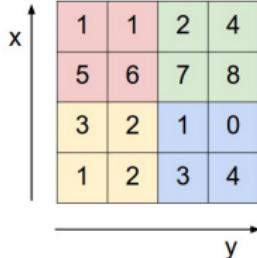
112x112x64



224  
224

downsampling

112  
112

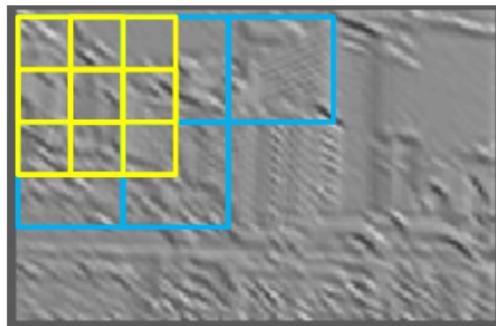


max pool with 2x2 filters  
and stride 2

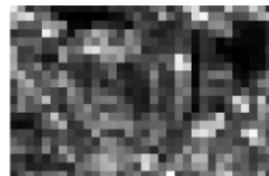
A 2x2 grid showing the result of max pooling. The values are:

6	8
3	4

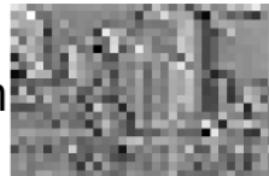
[CS231]



Max



Sum

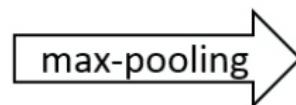


[Fergus, 2014]

# Rückwärtsgang bei Max-Pooling

0.1	0.5	<b>1.2</b>	-0.7
<b>0.8</b>	-0.2	-0.5	0.3
0.4	<b>0.9</b>	-0.1	-0.2
-0.6	0.1	<b>0.5</b>	0.3

Activations



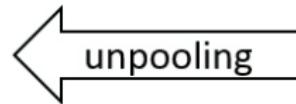
0.8	1.2
0.9	0.5

Mask

		X	
X			
	X		
		X	

max locations

0	0	<b>0.5</b>	0
<b>1.3</b>	0	0	0
0	<b>0.4</b>	0	0
0	0	<b>0.1</b>	0



Gradients

1.3	0.5
0.4	0.1

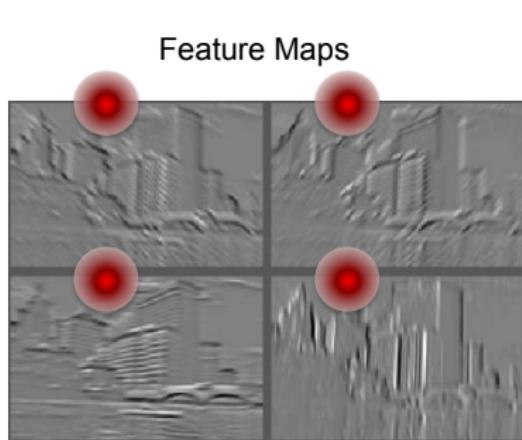
[<https://mukulrathi.com/demystifying-deep-learning/conv-net-backpropagation-maths-intuition-derivation/>]

Bei Durchschnittspooling wird der Fehler auf alle 4 Ursprungspixel mit dem Faktor  $\frac{1}{4}$  verteilt.

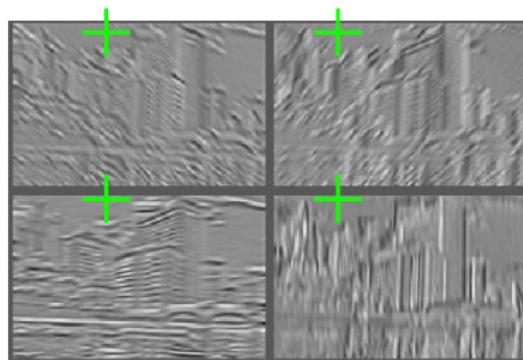
# Lokale Kontrastadaption

Die Aktivitäten der Faltungsschichten können sehr stark variieren. Oft ist es nützlich, eine *lokale Kontrastadaption* vorzunehmen, z.B. indem man alle zu einer Bildposition gehörigen Filterneuronen auf Varianz 1 normiert:

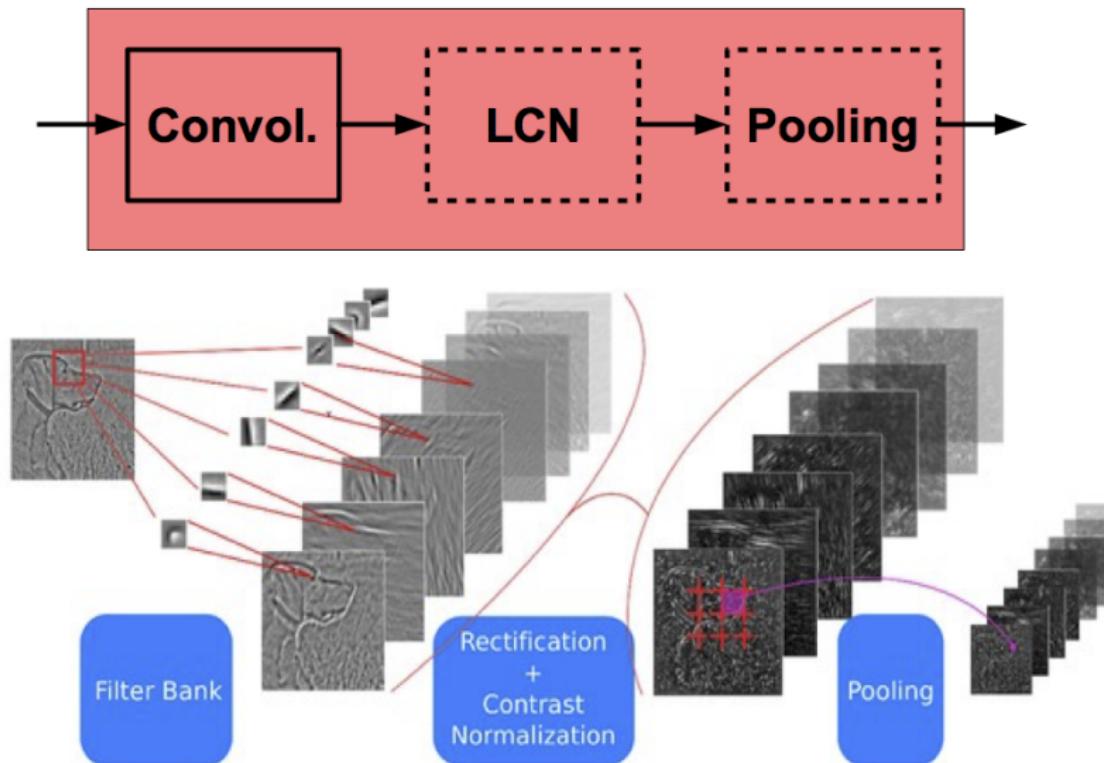
$$a_{x,y,z}^l = a_{x,y,z}^{l-1} / \sqrt{\frac{1}{p-1} \sum_z (a_{x,y,z}^{l-1})^2}.$$



Feature Maps  
After Contrast Normalization



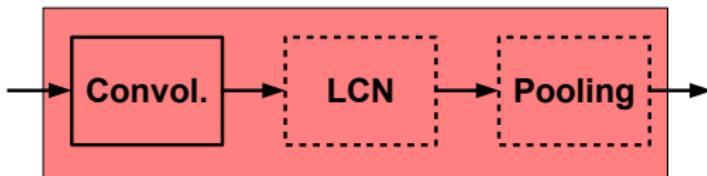
# Typische Stufe eines Faltungsnetzes



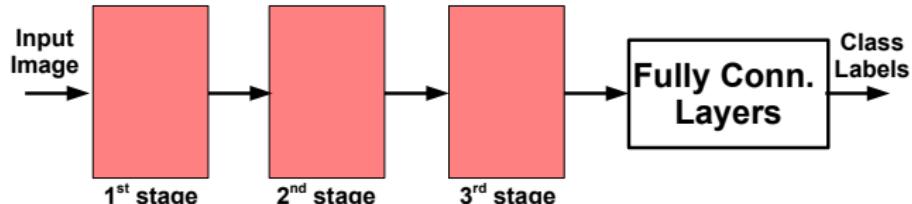
[Ranzato, 2014]

# Faltungsnetz: typische Architektur

One stage (zoom)

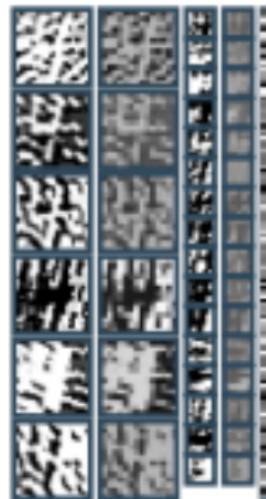


Whole system

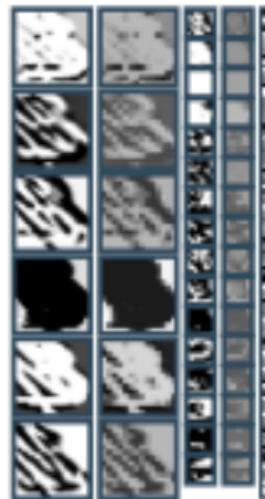


[Ranzato, 2014] Nach einigen Stufen bekommen die Neuronen Informationen aus dem gesamten Bild, d.h. nach dem Training haben wir **gelernte Deskriptoren**, die den gesamten Bildinhalt beschreiben!

# Beispiel: Faltungsnetz zur Zeichenerkennung



4



3



[LeCun et al., 1998]

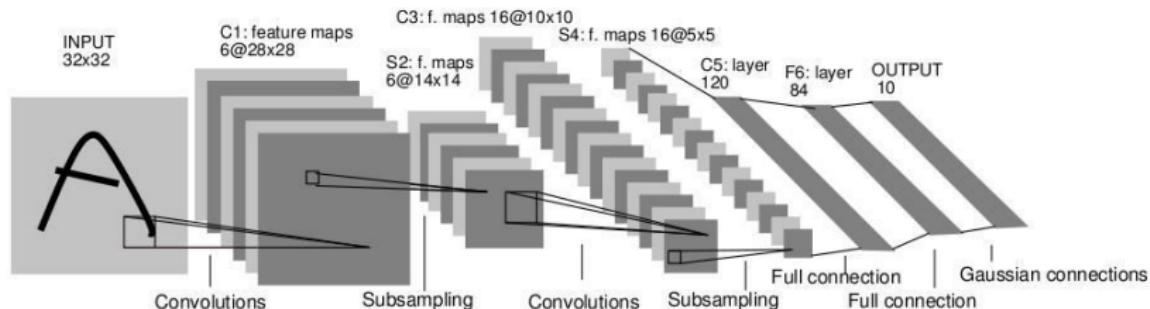
# Übersicht

1 Moderne CNN-Architekturen

2 Deep Learning mit Faltungsnetzen

3 Moderne CNN-Architekturen

# Urvater aller CNNs: LeNet 5 (LeCun, 1998)



- 0.8% Fehler auf MNIST-Datensatz
- 7 Schichten, davon 2 Faltungs- und 2 Pooling-Schichten
- 8.094 Neuronen, 344.068 Verknüpfungen, 11.880 frei trainierbare Parameter.
- 20 Iterationen über 65.000 Trainingsbeispiele, Gradientenabstieg mit stochastischem Levenberg-Marquart-Algorithmus

## LeNet Demo 1993



Läuft auf einem 486er-PC mit AT&T DSP32C add-on board (20 Mflops!)

# ImageNet

200 Klassen, 450.000 Bilder der Größe  $482 \times 415$

Beispiel Hammer:



# CNN-Architekturen der ImageNet-Gewinner

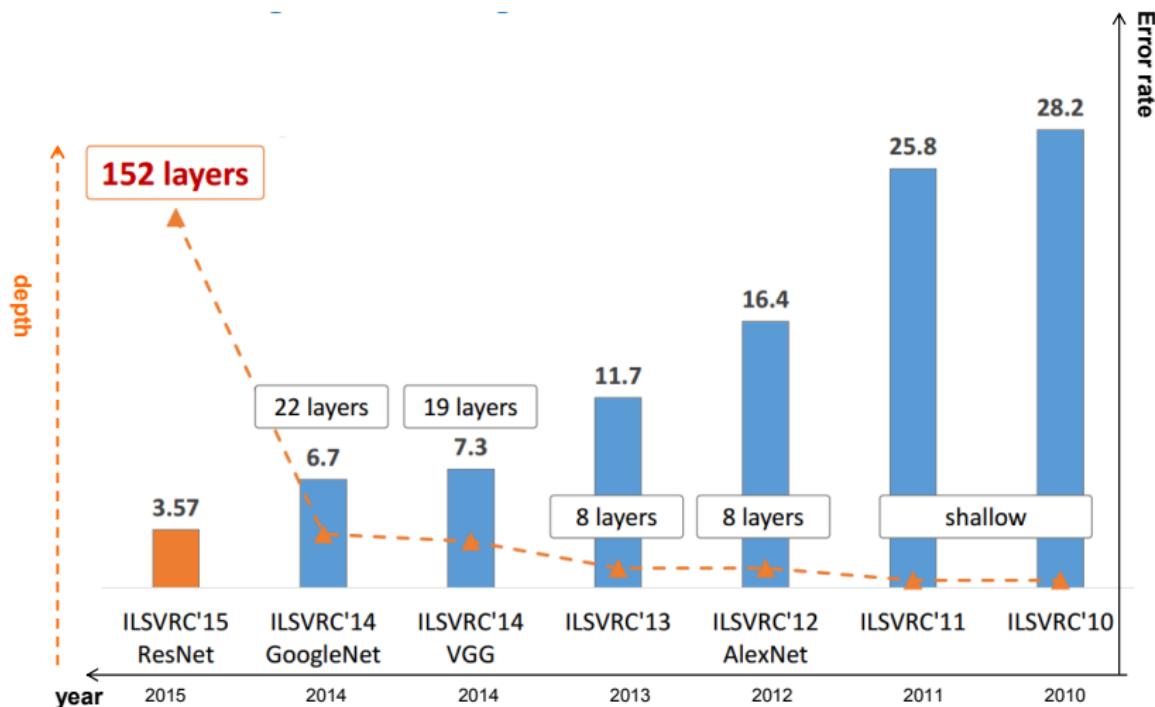
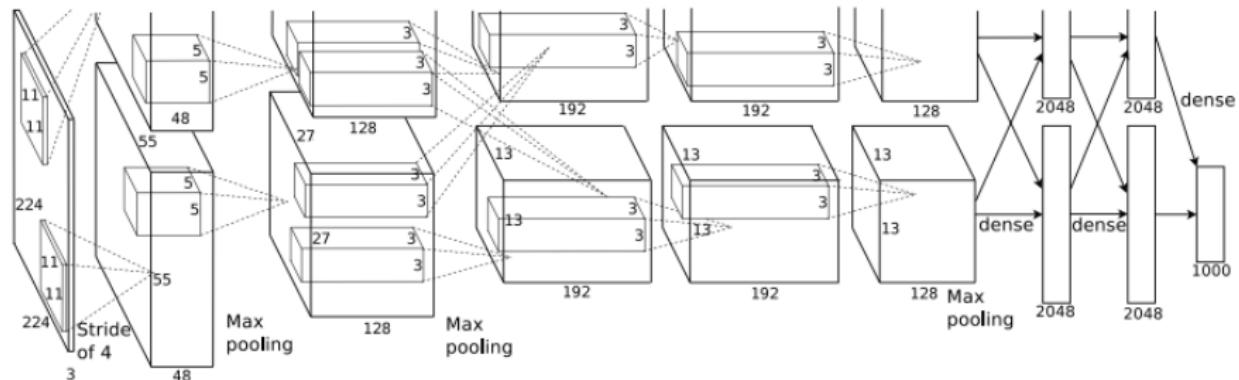


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. "Deep Residual Learning for Image Recognition". arXiv 2015

# AlexNet (Krishevsky et al., 2012)



- 8 Schichten, 650.000 Neuronen, 60 Mio. Verknüpfungen
- Trainiert eine Woche lang auf 2 GPU-Karten.
- ReLUs statt sigmoider Aktivierungsfunktionen.

# AlexNet: Durchbruch der Deep-Learning-Architekturen



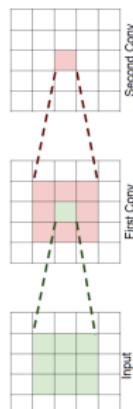
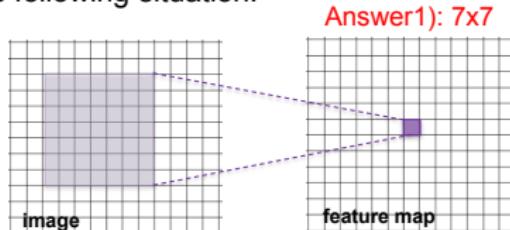
Gewinner der ImageNet 2012: AlexNet verbesserte den Fehler von 25.8% auf 16.4%. Danach wurden i.W. nur noch Deep-Learning-Architekturen verwendet.

# Trend zu kleineren Filtern und tieferen Architekturen

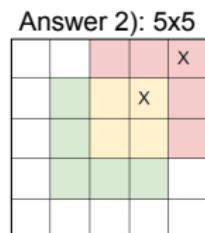
Why do modern architectures use very small filters?

Determine the receptive field in the following situation:

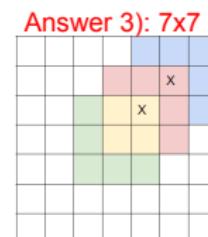
- 1) Suppose we have one  
7x7 conv layers (stride 1)  
49 weights



- 2) Suppose we stack two  
3x3 conv layers (stride 1)



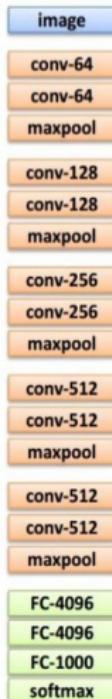
- 3) Suppose we stack three  
3x3 conv layers (stride 1)  
 $3 \times 9 = 27$  weights



We need less weights for the same receptive field when stacking small filters!

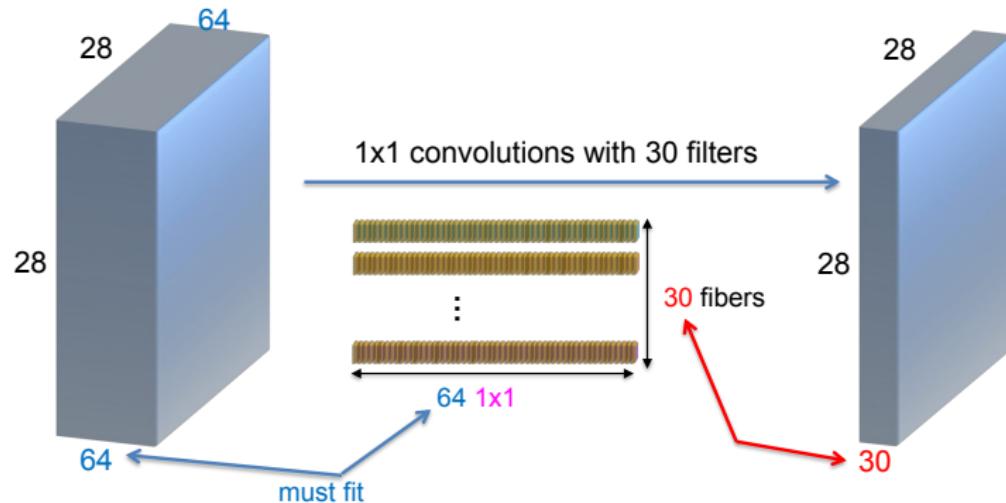
[B. Sick]

# ”Oxford Net” oder ”VGG19”: zweiter Platz 2014



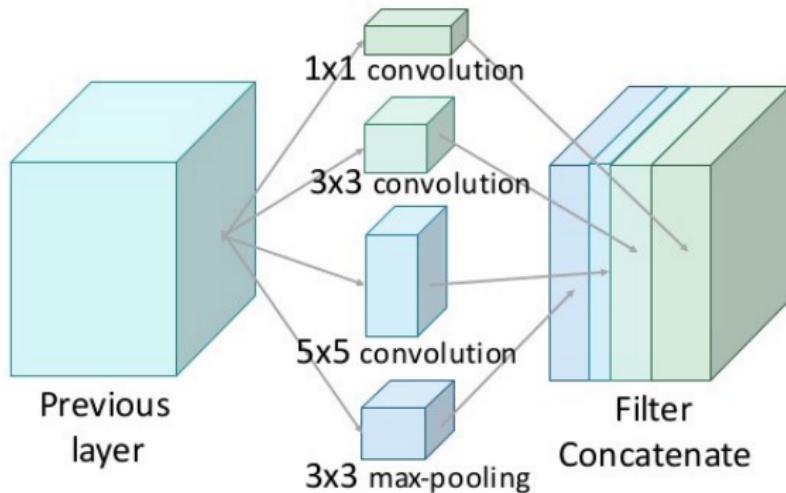
- 19 Schichten
- Mehrere 3x3-Faltungsschichten hintereinander vor Max-Pooling: große rezeptive Felder mit weniger freien Parametern.
- Kleine Poolingregionen
- Keine gleichzeitige Unterabtastung bei Faltung
- ReLU-Aktivierungsfunktion nach Faltungs- und voll verbundenen Schichten
- Vortrainierte Faltungsschichten sind heute Standard-Features für Bilderkennungsanwendungen.

## Filtergröße 1: 1x1-Faltungen wirken nur in die Tiefe



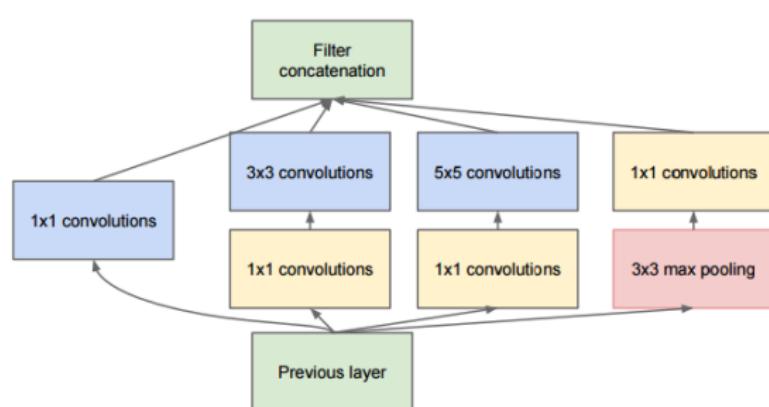
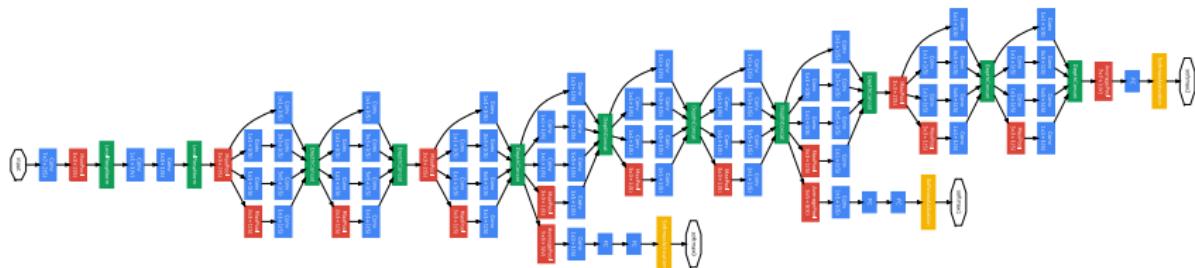
- 1x1-Faltungen wirken nur in der Tiefendimension entlang der verschiedenen Kanäle.
- Auf diese Weise kann die Anzahl der Kanäle verändert werden, ohne dass sich das Bild verändert.
- Führt zu einem verstärkt nichtlinearen Verhalten des Netzes.

# ”Inception”-Modul



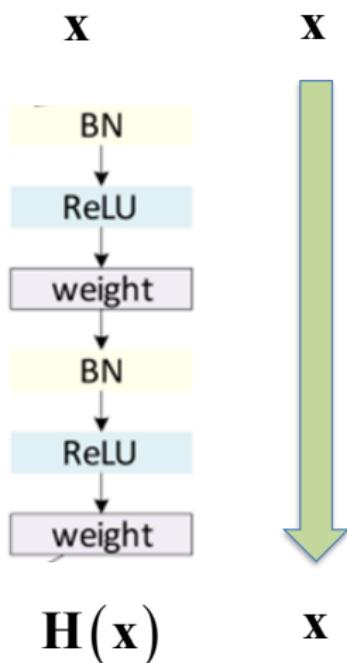
- Zwischen zwei Schichten werden mehrere Operationen **parallel** durchgeführt: 1x1-, 3x3-, 5x5-Faltungen und Max-Pooling.
- Die resultierenden Feature-Maps werden als neue Kanäle im Output hintereinander gestapelt.
- Im Vergleich zu Faltungen mit der gleichen Anzahl von Output-Kanälen braucht man weniger Gewichte.

# "GoogLeNet": Gewinner 2014



Inception-Modul mit  
Dimensionsredukti-  
on durch  
1x1-Faltungen.

# ”Highway”-Netzwerke: Je nach Input kann der Gradient Schichten überspringen



Idee: das Netzwerk kann lernen, bei bestimmten Inputs eine Schicht zu überspringen. Dies geschieht über eine Gate-Funktion  $T$ , die die gleiche Größe wie die Netzwerkschicht hat.

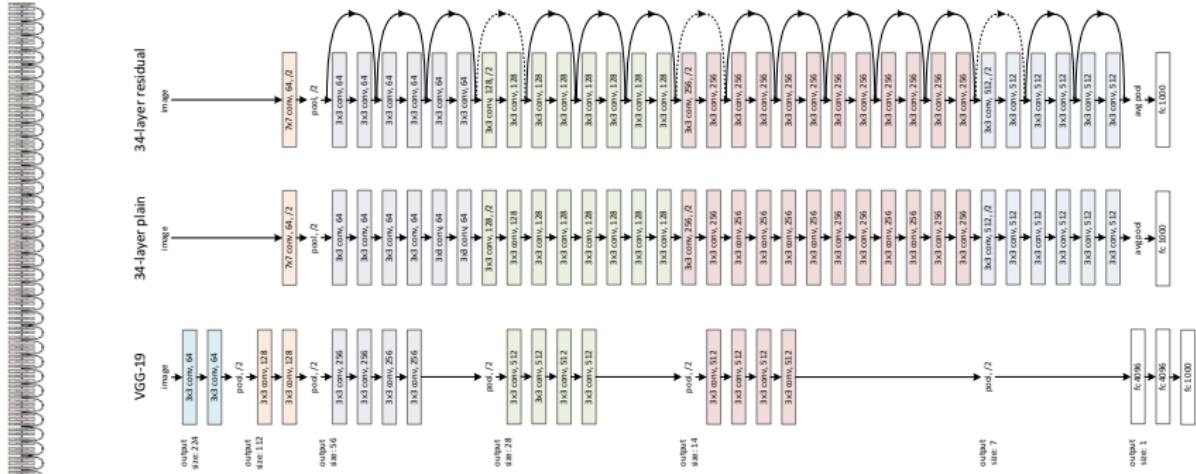
Netzwerkschicht ohne Gate (Input und Output haben dieselbe Größe, z.B. Faltungsschicht):

$$y = H(x, W_H)$$

Netzwerkschicht mit Gate  
 $T(x, W_T, b_T) = \sigma(W_T x + b_T) \in [0, 1]$  und lernbaren Gewichten  $W_T, b_T$ :

$$y^* = T(x, W_T, b_T) \odot y + (1 - T(x, W_T, b_T)) \odot x$$

# Microsoft "ResNet": Gewinner 2015



152 Schichten! Kann dennoch trainiert werden, weil der Gradient Schichten überspringen kann.

Architektur ist ähnlich zu VGG19, aber alle 2 Faltungsschichten gibt es eine mögliche Abkürzung.